
Proving Properties of Multidimensional Recurrences

David Cachera

Overview

- Motivations for formal verification
- Using MMAAlpha to prove simple properties
- An Alpha/PVS interface
- How can we benefit from the polyhedral model?

Formal verification

- Model checking: finite systems (or abstraction)
limited logic
automatic (succeeds or counterexample)
- Theorem proving: arbitrarily complex systems
higher order logic
requires user interaction

Proving... but why?

- MMAAlpha transformations generally ensure correctness (equivalence), but some refinement steps do not totally preserve the semantics
- Prove “partial” properties, or properties that are not expressible within the formalism of SAREs
- Complex, structured Alpha programs → errors!

Example: APP

```
system APP { N | 1<=N }
  ( a : { i,j | 0<i,j<=N } of integer )
  returns app : { i,j,k | 0<=k<=N; 0<i,j<=N } of integer
  app[i,j,k] = case
    { | k=0 } : a[i,j];
    { | k>0; i=j=k } : closure(app[i,j,k-1]);
    { | k>0; j=k; (i>k|i<k) } : app[i,k,k] * app[i,j,k-1];
    { | k>0; i=k; (j>k|j<k) } : app[i,j,k-1] * app[k,j,k];
    { | k>0; (i>k|i<k);(j>k|j<k) } :
      app[i,j,k-1] + app[i,k,k] * app[k,j,k-1];
  esac;
```

Proving within MMAAlpha

APP free schedule

```
free[i,j,k] = case
  { | k=0 } : 0;
  { | k>0; i=j=k } : 1 + free[i,j,k-1];
  { | k>0; j=k; i>k | i<k } :
    1 + (free[i,k,k] max free[i,j,k-1]);
  { | k>0; i=k; j>k | j<k } :
    1 + (free[i,j,k-1] max free[k,j,k]);
  { | k>0; i>k>j | j>k>i | j,i>k | i,j<k } :
    1 + (free[i,j,k-1] max free[i,k,k] max free[k,j,k]);
  esac;

closed[i,j,k] = case
  { | k=0 } : 0;
  { | k>0; i=j=k } : 3*k - 2;
  { | k>0; j=k; i>k | i<k } | { | k>0; i=k; j>k | j<k } : 3*k - 1;
  { | k>0; i>k>j | j>k>i | j,i>k | i,j<k } : 3*k;
  esac;
```

Proving within MMAAlpha

`free=exp(free)` `closed=f(i,j,k)`

- Write down the expression corresponding to
`Theorem=(closed=exp(closed))`
- If we prove that `Theorem` is true on its entire domain, we have proven that `free` and `closed` follow the same recursion scheme
- Meta argument based on the denotational semantics: if `free` and `closed` follow the same recursion scheme, then they are equal on their entire domain
- Normalization of the expression defining `Theorem`
 - set of tautologies (?)
 - PVS decision procedures

Proving within MMAAlpha

- Write down the equation

`Theorem = (free = closed)`

- Substitute `Free` by its defining expression

`Theorem = (exp(free) = closed)`

- Substitute all occurrences of `free` by `closed`

This step does not preserve semantics

`Theorem = (exp(closed) = closed)`

- Substitute all occurrences of `closed` by their defining expression

`Theorem = (exp(f(i,j,k)) = f(i,j,k))`

Proving within MMAAlpha

A non-linear schedule

```
t[i,j,k]= case
  { | k=0 } : 0[];
  { | k>0 } : (n[]+2[])*(k[]+1[]) + j[] * n[] + i[] + 1[];
esac;
X[i,j,k]= case
  { | k=0 } : True[];
  { | k>0; i=j=0 } : t[i,j,k] > t[i+1,j+1,k-1];
  { | k>0; i=0; 0<j<n-1 } : t[i,j,k] > max(t[i+1,j+1,k-1], t[0,0,k]);
  { | k>0; i=0; j=n-1 } : t[i,j,k] > max(t[i+1,0,k-1], t[0,0,k]);
  { | k>0; j=0; 0<i<n-1 } : t[i,j,k] > max(t[i+1,j+1,k-1], t[0,0,k]);
  { | k>0; j=0; i=n-1 } : t[i,j,k] > max(t[0,j+1,k-1], t[0,0,k]);
  { | k>0; n-1>j>0; n-1>i>0 } :
    t[i,j,k] > max(max(t[i+1,j+1,k-1], t[i,0,k]), t[1,j,k-1]);
  { | k>0; n-1=j; n-1>i>0 } :
    t[i,j,k] > max(max(t[i+1,0,k-1], t[i,0,k]), t[1,j,k-1]);
  { | k>0; n-1>j>0; n-1=i } :
    t[i,j,k] > max(max(t[0,j+1,k-1], t[i,0,k]), t[1,j,k-1]);
  { | k>0; n-1=j; n-1=i } :
    t[i,j,k] > max(max(t[0,0,k-1], t[i,0,k]), t[1,j,k-1]);
esac;
```

Why PVS?

More complex properties / More complicated proofs

Alpha and MMAAlpha:

- Only affine index expressions
- Only universally quantified formulas

PVS:

- Proofs more complex than just equivalence
- PVS induction schemes and higher order logic

PVS

- Specification language
 - collection of **theories**: assumptions, functions and predicate definitions, axioms and theorems
 - user-defined types, possibly unspecified
 - higher-order logic

PVS

- Prover

- derivation from one goal to one (or more) other(s)

$$\begin{array}{c} A_1 \\ A_2 \\ \dots \\ \hline B_1 \\ B_2 \\ \dots \end{array} \quad \begin{array}{c} \frac{A_1 \vee A_2}{B} \\ \swarrow \quad \searrow \\ \frac{A_1}{B} \quad \frac{A_2}{B} \end{array}$$

$A_1 \wedge A_2 \wedge \dots \Rightarrow B_1 \vee B_2 \vee \dots$

- user-guided derivations
- predefined and user-defined strategies (\rightarrow automation)

An Alpha/PVS interface

Translates Alpha systems into PVS theories.
Runs PVS with adapted strategies.

Classification of properties

- “axioms”: input variables
- “simple” properties: do not need PVS induction to be proved
- “inductive” properties: have to be proven by means of an induction scheme

An Alpha/PVS interface

Equations and variables

Equational rather than functional point of view.

- “Concrete” variable: translated as PVS axiom, relation between the variable and its defining expression
- “Abstract” variable: unspecified mapping from domains to values. Behaviour specified by axioms

An Alpha/PVS interface

Proof strategies

- Try to determine which properties or axioms are necessary
- Avoid getting into infinite rewriting loops
- Make use of PVS induction schemes

Modular development

- Use of Alpha subsystems
- Mechanism of abstract variables

An example of a structured proof

```
--include MatVect.alpha
system MatMat : {N,M,P | N,M,P > 1}
  ( A :{i,j|1 <= i <= N; 1 <= j <=M} of real;
    B :{i,j|1 <= i <= M; 1 <= j <=P} of real)
  returns ( C: {i,j|1 <= i <= N; 1 <= j <=P} of real);
let
  use {k|1 <= k <= P} MatVect [N,M] (A.(i,j,k->i,j), B) returns (C);
tel;

system MatVect : {N,M | N,M > 0}
  ( A : {i,j|1 <= i <= N; 1 <= j <= M} of real;
    b : {i|1 <= i <= M} of real)
  returns ( c : {i|1 <= i <= N} of real);
var C : {i,j|1 <= i <= N; 0 <= j <= M} of real;
let
  C[i,j] = case
    {j=0}   : 0[];
    {j>=1} : C[i,j-1] + A[i,j]*b[j];
  esac;
  c[i] = C[i,M];
tel;
```

An example of a structured proof

```
"axiom"["FORALL (i,j,N,M : int ) : (-1 + i >= 0 AND -1 + j >= 0
                                     AND -i + N >= 0 AND -j + M >= 0)
      IMPLIES MatVect_A(i,j,N,M)>=0"],
```

```
"axiom"["FORALL (i,N,M : int ) : (-1 + i >= 0 AND -i + M >= 0 )
      IMPLIES MatVect_b(i,N,M)>=0"],
```

```
"inductprop"["FORALL (j:nat) : FORALL (i,N,M : int ) : (-1 + i >= 0
      AND -i + N >= 0 AND -j + M >= 0 AND N>=1 AND M>=1)
      IMPLIES MatVect_C(i,j,N,M)>=0", "j"],
```

```
"simpleprop"["FORALL (i,N,M : int ) : (-1 + i >= 0 AND -i + N >= 0
      AND M>=1 AND N>=1)
      IMPLIES MatVect_c(i,N,M)>=0"]
```

Polyhedral model

- PVS translation: doesn't take into account the specificities of the model
- Control properties
- More polyhedra manipulations

Et après ?

Ce qu'il ne faut pas faire : vérification de SoC

Formaliser le raffinement : comment exprimer/valider les transformations ?

Modèle : futur de MMAAlpha ?

Extension ? Modèles hybrides ?

Et après ?

Les méthodes formelles, ça n'est pas que la vérification

Flot de compilation complexe

- Comment s'articulent les différents "modules" ?
- Formalisation des interfaces ?
- Modèles de coût ? (interprétation abstraite ?)